Booking up for Beauty

Welcome to Booking up for Beauty on Exercism's Go Track. If you need help running the tests or submitting your code, check out `HELP.md`. If you get stuck on the exercise, check out `HINTS.md`, but try and solve it without using those first :)

Introduction

A `Time` in Go is a type describing a moment in time. The date and time information can be accessed, compared, and manipulated through its methods, but there are also some functions called on the `time` package itself. The current date and time can be retrieved through the `time.Now` function.

The `time.Parse` function parses strings into values of type `Time`. Go has a special way of how you define the layout you expect for the parsing. You need to write an example of the layout using the values from this special timestamp: `Mon Jan 2 15:04:05 -0700 MST 2006`.

For example:

```
import "time"

func parseTime() time.Time {
    date := "Tue, 09/22/1995, 13:00"
    layout := "Mon, 01/02/2006, 15:04"

    t, err := time.Parse(layout,date) // time.Time, error
}

// => 1995-09-22 13:00:00 +0000 UTC
```

The `Time.Format()` method returns a string representation of time. Just as with the `Parse` function, the target layout is again defined via an example that uses the values from the special timestamp.

For Example:

```
import (
    "fmt"
    "time"
)

func main() {
    t := time.Date(1995,time.September,22,13,0,0,0,time.UTC)
    formatedTime := t.Format("Mon, 01/02/2006, 15:04") // string
    fmt.Println(formatedTime)
}

// => Fri, 09/22/1995, 13:00
```

Layout Options

For a custom layout use combination of these options. In Go predefined date and timestamp format constants are also available.

Time	Options
Year	2006 ; 06
Month	Jan ; January ; 01 ; 1
Day	02 ; 2 ; _2 (For preceding 0)
Weekday	Mon ; Monday
Hour	15 (24 hour time format) ; 3 ; 03 (AM or PM)
Minute	04 ; 4
Second	05 ; 5
AM/PM Mark	PM
Day of Year	002 ; __2

The `time.Time` type has various methods for accessing a particular time. e.g. Hour : `Time.Hour()` , Month : `Time.Month()` . More on how this works can be found in official documentation.

The `time` includes another type, `Duration` , representing elapsed time, plus support for locations/time zones, timers, and other related functionality that will be covered in another

concept.

Instructions

In this exercise you'll be working on an appointment scheduler for a beauty salon that opened on September 15th in 2012.

You have five tasks, which will all involve appointment dates.

1. Parse appointment date

Implement the `Schedule` function to parse a textual representation of an appointment date into the corresponding `time.Time` format:

```
Schedule("7/25/2019 13:45:00")
// => 2019-07-25 13:45:00 +0000 UTC
```

2. Check if an appointment has already passed

Implement the `HasPassed` function that takes an appointment date and checks if the appointment was somewhere in the past:

```
HasPassed("July 25, 2019 13:45:00")
// => true
```

3. Check if appointment is in the afternoon

Implement the `IsAfternoonAppointment` function that takes an appointment date and checks if the appointment is in the afternoon (>= 12:00 and < 18:00):

```
IsAfternoonAppointment("Thursday, July 25, 2019 13:45:00")
// => true
```

4. Describe the time and date of the appointment

Implement the `Description` function that takes an appointment date and returns a description of that date and time:

```
Description("7/25/2019 13:45:00")
// => "You have an appointment on Thursday, July 25, 2019, at 13:45."
```

5. Return the anniversary date of the salon's opening

Implement the `AnniversaryDate` function that returns the anniversary date of the salon's opening for the current year in UTC.

Assuming the current year is 2020:

```
AnniversaryDate()

// => 2020-09-15 00:00:00 +0000 UTC
```

Note: the return value is a `time.Time` and the time of day doesn't matter.

Source

Created by

- @jamessouth